Interoperable Semantic & Syntactic Service Matching for Ambient Computing Environments

Sonia Ben Mokhtar[3], Pierre-Guillaume Raverdy[1], Aitor Urbieta[2], Roberto Speicys Cardoso[1]

[1] INRIA-Rocquencourt, firstname.lastname@inria.fr
[2] Mondragon Unibertsitatea Aurbieta@eps.mondragon.edu
[3] University College London s.benmokhtar@cs.ucl.ac.uk

Abstract. The inherent heterogeneity of ambient computing environments and their constant evolution requires middleware platforms to manage networked components designed, developed and deployed independently. Such management must also be efficient to cater for resource-constrained devices and highly dynamic situations due to the spontaneous appearance and disappearance of networked resources. For service discovery protocols (SDP), one of the main functions of service-oriented architectures (SOA), the efficiency of the matching of syntactic service descriptions is most often opposed to the fullness of the semantic approach. As part of the PLASTIC middleware for ambient computing environments, we present in this paper an interoperable service matching and ranking platform, which is able to process service descriptions from both semantic and syntactic service description languages. To that end, we define a generic, modular description language able to record service functional properties, potentially extended with semantic annotations. An evaluation of the prototype implementation of our platform demonstrates that multi-protocols service matching supporting various levels of expressiveness can be achieved in ambient computing environments.

Keywords: Service-oriented ambient computing, service discovery, matching and ranking, interoperability, semantic-awareness.

1 Introduction

Ambient computing envisions the unobtrusive diffusion of computing and networking resources in physical environments, enabling users to access information and computational resources anytime and anywhere, and this in a user-centric way, i.e., where user interaction with the system is intuitive, pleasant and natural. Mobile users take part in these ambient computing environments by carrying around tiny personal devices that integrate seamlessly in the existing infrastructure. Such a setup is highly open and dynamic. Therefore, these environments must support the dynamic deployment and execution, integrating the available hardware and software resources at any given time and place. This dynamic merging is facilitated when organizing resources as autonomous, networked components. Towards this purpose, the Service-Oriented Architecture (SOA) computing paradigm is particularly appropriate for ambient computing systems. Indeed, in this architectural style, networked devices and

their hosted applications are abstracted as loosely coupled services that can be integrated into larger systems. Service discovery (SD) is then an essential function within SOA as it enables the runtime association to networked services. Specifically, to feature the dynamics of ambient computing environments, which is characterized by the spontaneous appearance and disappearance of heterogeneous networked services, SD must at once address a wide range of interoperability issues such as multi-protocols SD and semantics.

Many research projects presented in Section 2 have addressed one of the above issues. In the context of the PLASTIC project[1], which aims to support the deployment and dynamic composition of mobile, adaptable applications in ambient computing environments, and in particular applications complying with Web-service standards, the PerSeSyn (*Pervasive Semantic Syntactic*) service discovery platform aims to address both multi-protocols and semantic interoperability issues in ambient computing environments. In this paper, we focus in particular on the challenges posed by enabling interoperable service matching dealing with both syntactic and semantic-based SDPs. Towards this purpose, we introduce in Section 3 the *PerSeSyn Service Description Model* (PSDM), and its instantiation, the *PerSeSyn Service Description Language* (PSDL). PSDM is a conceptual model for enabling semantic mapping between heterogeneous service description languages. PSDL, which is an instantiation of PSDM, is not yet another service description language but a combination of emergent standards for service specification, namely SAWSDL[2] and WS-BPEL[3]. PSDL is then employed as the common representation for service descriptions and requests. Based on PSDM and PSDL, we define a set of conformance relations, as presented in Section 4, for matching heterogeneous service descriptions going from elementary syntactic service descriptions (e.g., given in SLP[4]) to rich semantic service descriptions with associated conversations (e.g., given in OWL-S[7]). Furthermore, we introduce a mechanism for ranking heterogeneous matching results towards effective matching of service capabilities. In order to achieve the efficiency required for resource-constrained devices, we assess semantic service matching using optimized mechanisms previously presented in [8]. We further evaluate in Section 5 the impact of introducing semantic based matching in addition to protocol interoperability realized through protocol translation. We finally summarize our contribution in Section 6.

2 Related Work

Service discovery protocols enable services on a network to discover each other, express opportunities for collaboration, and compose themselves into larger collections that cooperate to meet an application's needs. Many academic and

[1] IST PLASTIC Project. http://www.ist-plastic.org/
[2] SAWSDL: Semantic Annotations for WSDL. http://www.w3.org/2002/ws/sawsdl/
[3] WS-BPEL: Web Services Business Process Execution Language. http://www.oasis-open.org/
[4] SLP: Service Location Protocol. http://www.ietf.org/rfc/rfc2608.txt

industry-supported SDPs have already been proposed such as UDDI[5] or CORBA's Trading Service[6] for the Internet, or SLP and Jini for local and ad hoc networks. Although many SDPs solutions with well-proven protocol implementations are now available, a number of challenging issues remain for the ambient computing environment. Among these issues multi-protocols service discovery and semantic-awareness are described below.

Multi-protocols SD. Middleware heterogeneity raises interoperability issues between the different SDPs (e.g., SLP, SSDP[7], UDDI) active in the environment. Existing SDPs do not directly interoperate with each other as they employ incompatible formats and protocols for service descriptions or discovery requests, and also use incompatible data types or communication models. Several projects have thus investigated interoperability solutions [3, 1, 4], as requiring clients and service providers to support multiple SDPs is not realistic. SDP interoperability is typically achieved using intermediate common representations of service discovery elements (e.g., service description, discovery request) [5] instead of direct mappings [4], as the latter does not scale well with the number of supported protocols. Furthermore, the interoperability layer may be located close to the network layer [5], and efficiently and transparently translate network messages between protocols, or may provide an explicit interface [2] to clients or services so as to extend existing protocols with advanced features such as context management. While the above solutions deal with multi-protocol heterogeneity they suffer from a common limitation, which is the *syntactic heterogeneity* of service descriptions. A promising approach towards addressing syntactic heterogeneity relies on semantic modeling of the services' functional features.

Semantic SD. The matching of service requests and service advertisements is classically based on assessing the syntactic conformance of service functional properties. However, an agreement on a common syntactic standard is hardly achievable in open environments. Thus, higher-level abstractions, independent of the low-level syntactic realizations specific to the technologies in use, should be employed for denoting service semantics. A number of approaches for semantic service specification have been proposed, and in particular for semantic Web services such as OWL-S, WSMO[8] or SAWSDL. OWL-S and WSMO are two ontologies for semantic service specification, while SAWSDL annotates Web services with semantics by attaching references to concepts from ontologies to WSDL[9] input, output and fault messages, as well as to operations. EASY [8] provides efficient semantic service discovery, a key requirement for the resource-limited devices found in ambient computing environments, by encoding ontology concepts off-line and adequately classifying service descriptions in the repository based on these encoded concepts.

[5] UDDI: Universal Description Discovery and Integration. http://uddi.xml.org/
[6] CORBA Trading Service. http://www.omg.org/
[7] SSDP: Simple Service Discovery Protocol. http://tools.ietf.org/html/draft-cai-ssdp-v1-03
[8] WSMO: Web Service Modeling Ontology. http://www.wsmo.org/
[9] WSDL: Web Service Description Language. http://www.w3.org/TR/wsdl

While existing research efforts deal with one of the above issues separately, a comprehensive solution for SD in ambient computing environments needs to support both multi-protocols and semantic-based interoperability. Multi-protocol, semantic-enhanced service discovery calls for interoperable service specification, matching and ranking as presented in sections 3, 4 and 5 respectively.

3 PerSeSyn Service Specification

The *PerSeSyn Service Description Model* (PSDM) serves as a basis for enabling mapping between heterogeneous service description languages including both syntactic and semantic-based languages. Specifically, service descriptions given using syntactic-based languages (e.g., UPnP[10], SLP, WSDL) and semantic-based languages (e.g., SAWSDL, OWL-S, WSMO) are translated to PSDL descriptions, which is an instantiation of the PSDM model presented in Section 3.2. As a result, a service request may be matched using any of the existing descriptions stored in the repository, independently of the underlying original service description language.

3.1 PSDM: Model for Semantic and Syntactic Service Specification

The design of PSDM builds upon two previous models: (1) the MUSDAC service model [6], which supports interoperability between syntactic based languages (e.g., UPnP, SLP and WSDL) and (2) the EASY service model [8] for the semantic specification of service functional and non-functional capabilities. In this paper, due to a lack of space, we do not present features related with the specification and matching of service non-functional properties.
The UML diagram depicted in Figure 1 shows the PSDM conceptual model. This model introduces the main conceptual elements that serve as a basis for matching service advertisements with service requests as further discussed in Section 4. Using this model, a service description is composed of two parts: a *profile*, and a *grounding*. The service profile is described as a non empty set of *capabilities* while the service grounding prescribes the way of accessing the service as described in the original legacy service description. A service capability is any functionality that may be provided by a service and sought by a client. It is described with its *Name* potentially associated with a *semantic annotation*, a possibly empty set of *inputs*, a possibly empty set of *outputs* and a potential *conversation*. Capabilities that do not have any associated input/output descriptions are those provided by legacy protocols that use names to characterize capabilities (e.g., a native SLP service). Inputs associated with a capability are the information necessary for the execution of the capability, while outputs correspond to the information produced by the capability. Input and outputs of capabilities are described with their *names*, *types* and a possible *semantic annotation* that is a reference to a concept in an existing ontology. Capabilities that do not have semantic annotations on the description of their inputs and outputs are those provided

[10] UPnP: Universal Plug and Play. www.upnp.org/

by legacy services without an enriched semantic interface (e.g., a native UPnP service). A conversation associated with a capability prescribes the way of realizing this capability through the execution of other capabilities organized in a workflow. Conversations are said to be *semantic* if they involve capabilities that have associated semantic annotations. They are said to be *syntactic* otherwise.
A capability that does not have an associated conversation is said to be *atomic*. Such capabilities correspond to service operations.

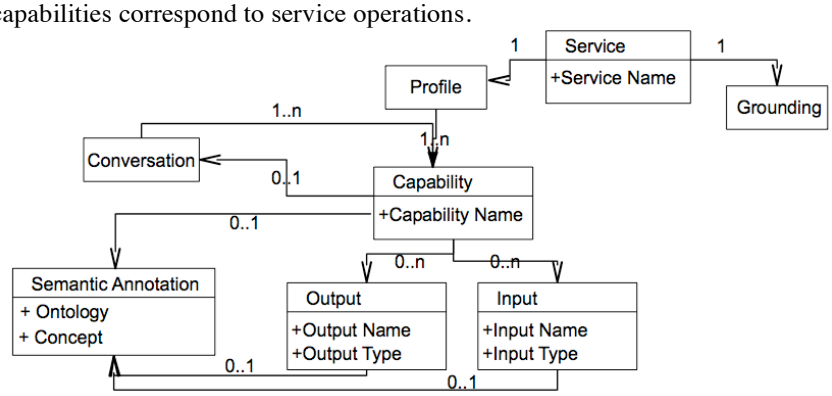

Fig. 1. PSDM

3.2 PSDL: Language for Semantic and Syntactic Service Specification

We define in this section the *PerSeSyn Service Description Language* (PSDL), a concrete realization of the PSDM model. This language is used as a common representation of service descriptions used for enabling interoperable service matching and ranking. For the implementation of PSDL, we opted for an XML-based schema defining a container, which is combined with the two emergent standard service description languages namely SAWSDL and WS-BPEL. The PSDL description acts primarily as a top-level container for additional files describing facets of the service. SAWSDL is used to describe the capability interfaces, while WS-BPEL is used to express conversations associated with capabilities. We employ SAWSDL for the definition of capability interfaces, as it supports both semantic and syntactic specification of service attributes (e.g., inputs, outputs). Thus, both legacy syntactic descriptions and rich semantic descriptions can be translated to SAWSDL. On the other hand, WS-BPEL is a comprehensive language for workflow specification, which is adequate for conversation specification. It has largely been adopted both in the industrial community and in academia. WS-BPEL supports only syntactic conversation specification, however, if combined with SAWSDL, semantic conversations can be defined. Furthermore, for the automated translation of service conversations we envision to build on the formal semantics of the existing languages (e.g., WS-BPEL [10], OWL-S [11]).

4 Matching and Ranking in PerSeSyn

Finding a service that exactly matches a client request is rather the exception than the rule in ambient computing environments. Thus, matching should be able to identify various degrees of conformance between services and clients, and rate services with respect to their suitability for a specific client request. We now present our set of conformance relations enabling interoperable matching of service capabilities in Section 4.1, and the associated service ranking mechanism in Section 4.2.

4.1 Interoperable Matching of Service Capabilities

Based on the PSDL language presented in Section 3, we present in this section a set of conformance relations for matching services in terms of their functional properties. One of the particular features of PSDL is the support of heterogeneous service description languages. Hence, the matching function used to assess the conformance of a service advertisement with a service request depends on the information contained in the request and the advertisement. For instance, comparing a service request described as a syntactic capability name (e.g., an SLP service) with a rich semantic service description (e.g., an SAWDL service), requires ignoring the semantic service annotations as well as input and output information and performing a syntactic comparison of the request with the capability names. The different cases of matching of heterogeneous service descriptions are outlined in Table 1. In this table a service request and a service advertisement can be described as: (1) a syntactic capability name; (2) a list of syntactic capabilities; (3) a list of semantic capabilities. Additionally, service advertisements can be further described as (4) a syntactic capability with an associated syntactic conversations and (5) a semantic capability with an associated semantic conversation. In this paper we do not consider the case where requests are specified with associated conversations. This may lead to a service composition process involving heterogeneous service advertisements, which we aim at addressing in our future work. The combination of the former scenarios gives fifteen cases for matching a service request with a service advertisement. Among these cases six cases are redundant in the table, which leads us to introduce the following six matching algorithms:

1. Syntactic matching of capability names, noted **SynNameMatch**(), consists of syntactically comparing the names of capabilities. However, as the syntactic matching has significant limitations such as false positive and false negative answers we rely on the function defined in [9] that uses the Wordnet[11] dictionary to evaluate the degree of similarity between two words of the dictionary. This similarity measure has a value in the interval [0,1]. Using this function, we define the function **SyntacticDistance** = *1-Wordnet similarity measure*, which we use with a threshold $\alpha \in [0,1]$ such that matching with distances above α are considered as failures. For instance, α=0 implies that a matching holds only if the two words are synonyms with respect to the Wordnet dictionary.

[11] Wordnet dictionary: http://wordnet.princeton.edu/

SynNameMatch() matching applies when either the request or the service are described as a syntactic capability name without an associated conversation (e.g., a SLP request with a SAWSDL service, a UPnP request with a SLP service). This function is defined as follows:

SynNameMatch(C_{adv},C_{req})=
SyntacticDistance(C_{adv}.*CapabilityName*,C_{req}.*CapabilityName*) ≤ α

2. Syntactic matching of capability names and conversation execution, noted **SynNameMatch+ExeConv**(), applies when the request is defined as a syntactic capability name and the service is either syntactic or semantic and has an associated conversation (e.g., an SLP request matched with an OWL-S service). In this case, we use the **SynNameMatch**() function defined above and if the matching holds the client executes the conversation associated with the matched capability.
3. Syntactic signature matching, noted **SynSigMatch**(), applies when the request is given as a list of syntactic capabilities and services are given as either a list of semantic capabilities or a list of syntactic capabilities. It applies also when a list of semantic capabilities is requested and only syntactic capabilities are provided by networked services. In the case of matching semantic capabilities with syntactic ones, the semantic annotations associated with capabilities' inputs and outputs are ignored. The **SynSigMatch**() function is defined as follows:

SynSigMatch(C_{adv},C_{req})=
∀ in ∈ C_{adv}.*In*, ∃*in'*∈ C_{req}.*In:* **SyntacticDistance**(*in.Name*,*in'.Name*) ≤ α and
∀ out ∈ C_{req}.*Out*, ∃*out'*∈ C_{adv}.*Out:* **SyntacticDistance**(*out.Name*,*out'.Name*) ≤ α

4. Syntactic signature matching and conversation execution, noted **SynSigMatch+ExeConv**(), applies when a service capability has an associated conversation either syntactically or semantically specified and the request is described as a list of capabilities, excluding the case where both of the request and the service are semantic-aware. In this case, we apply the **SynSigMatch**() matching defined above to assess the conformance of each requested capability with the provided ones, and if the matching holds, the client executes the associated conversation.
5. Semantic signature matching, noted **SemSigMatch**(), applies only when both the request and the advertisement are described as a set of semantic capabilities. While we previously defined a function for semantically matching service capabilities, this matching only supports fully semantically annotated capabilities [8]. We thus improve this function with the support of hybrid matching of service capabilities. Hybrid matching applies when services are not fully annotated, i.e., they may have both syntactic and semantic attributes, and is defined as follows:

SemSigMatch(C_{adv},C_{req})=
∀ in ∈ C_{adv}.*In*, ∃*in'*∈ C_{req}.*In:*

$$\begin{cases} \textit{if } (in.SemanticAnnotation \neq null \ \&\ in'.SemanticAnnotation \neq null) \\ \textbf{SemanticDistance}(in,in') \leq \beta \ \textit{Else} \\ \textbf{SyntacticDistance}(in.Name,in'.Name) \leq \alpha \end{cases}$$

and
∀ out ∈ C_{req}.*Out*, ∃*out'*∈ C_{adv}.*Out*

$$\begin{cases} \textit{if } (out.SemanticAnnotation \neq null \ \&\ out'.SemanticAnnotation \neq null) \\ \textbf{SemanticDistance}(out,out') \leq \beta \ \textit{Else} \\ \textbf{SyntacticDistance}(out.Name,out'.Name) \leq \alpha \end{cases}$$

Where **SemanticDistance**() is a function used to check whether two concepts are related in an ontology (i.e., if one is more generic than the other) and returns the number of levels that separate these two concepts in the ontology hierarchy. We also use this function with a threshold β, which indicates the maximal number of levels that separate two concepts in an ontology above which the matching is considered as a failure.

6. Semantic signature matching and conversation execution, noted **SemSigMatch+ExeConv**(), applies in the only case where a request described as a set of semantic capabilities is matched with a service described as a semantic capability with an associated conversation. In this case, we use the **SemSigMatch**() function to assess the conformance between semantic capabilities and the client has to execute the service conversation.

Table 1. Interoperable Matching of Services Capabilities

		Request		
		Syntactic capability name	**List of syntactic capabilities**	**List of semantic capabilities**
Service	**Syntactic capability name**	SynNameMatch	SynNameMatch	SynNameMatch
	List of syntactic capabilities	SynNameMatch	SynSigMatch	SynSigMatch
	List of semantic capabilities	SynNameMatch	SynSigMatch	SemSigMatch
	Syntactic conversation	SynNameMatch + ExeConv	SynSigMatch + ExeConv	SynSigMatch + ExeConv
	Semantic conversation	SynNameMatch + ExeConv	SynSigMatch + ExeConv	SemSigMatch + ExeConv

4.2 Ranking Heterogeneous Matching Results

We present in this section our mechanism for ranking service advertisements with respect to a service request. First, according to the degree of expressiveness of the service request, results coming from the various matching algorithms employed to assess the conformance of a service advertisement with the request, are ranked according to Table 2. For instance, for a request described as a list of semantic capabilities, ranking is performed according to the following expression:

SemSigMatch() > SemSigMatch+ExeConv() > SynSigMatch() > SynSigMatch+ExeConv() > SynNameMatch() > SynNameMatch+ExeConv()

This means that results of semantic matching functions are preferred to results of syntactic matching functions. Furthermore, capabilities that do not have associated conversations are preferred to capabilities that require the execution of the corresponding service conversation by the client. Finally, results coming from the weakest matching function (i.e., syntactic matching of capability names) are given the least scores compared to the others.

Table 2. Ranking Heterogeneous Matching Results

Request type	**Preferred matching functions**
Syntactic capability name	*SynNameMatch* > *SynNameMatch+ExeConv*
List of syntactic capabilities	*SynSigMatch* > *SynSigMatch+ExeConv* > *SynNameMatch* > *SynNameMatch+ExeConv*
List of semantic capabilities	*SemSigMatch* > *SemSigMatch+ExeConv* > *SynSigMatch* > *SynSigMatch+ExeConv* > *SynNameMatch* > *SynNameMatch+ExeConv*

On the other hand, the results of each of these matching functions are themselves classified according to their degree of conformance to the given request. The degree of conformance between a service request and a service advertisement is evaluated using the function **ServiceDistance**() which sums the results of **SyntacticDistance**() and **SemanticDistance**() functions used in the matching phase. If many provided capabilities have the same degree of conformance to a requested capability we envision using non-functional properties (e.g., QoS) to select the most appropriate one [8].

5 Prototype Implementation and Performance Evaluation

We have implemented a prototype of the PerSeSyn discovery platform using Java 1.6. To evaluate the efficiency of PerSeSyn, we evaluate the processing time to create PSDL requests and descriptions as a result of the translation of a legacy request/description. We also evaluate the processing time of matching various combinations of requests and descriptions. Tests are performed on a Windows XP PC with a 2.6GHz processor and 512 MB of memory. Results presented below are the average of 1000 tests. The standard deviation for the results presented below is negligible (less than 1%). As presented in [2], providing interoperability on top of simple, limited SDPs such as SLP may incur a significant overhead (i.e., overhead of over 200 milliseconds for a native discovery time of less than 1 millisecond for a similar configuration). It was analyzed that this overhead was by and large (two-thirds or almost 140 milliseconds) triggered by the SOAP-based interface of the interoperability service. This overhead however becomes negligible when interoperating with other SDPs such as UDDI that have a native discovery time between 1 and 6 seconds.

Table 3. Legacy to PSDL translation (micro-seconds)

	SLP	UPnP	WSDL
Discovery request	22.8	32.4	243
Discovery Request + Translation to PSDL	23.4	85.1	287
Overhead of the Translation	0.6	52.7	44

For the PerSeSyn prototype, the processing time for the translation of service descriptions (requests and advertisements) from selected legacy SDPs to PSDL descriptions are provided in Table 3. The first line of this table represents the time to process a discovery request using SLP, UPnP and WSDL excluding the time to parse

XML descriptions. The second line represents the time to process a discovery request in addition to the time to translate the request to PSDL. Finally, the third line represents the overhead of the translation. In this experiment times are given in microseconds. As it can be observed, this time increases with the complexity of the original description, and in particular the complexity and size of the original XML data to process. Overall, the translation time is not significant (tens to hundreds of micro-seconds) compared to the overall discovery time.

In Table 4, the processing time of the matching algorithms for the different combination of service requests and advertisements are provided. From the results of this experiment we can notice that the time to parse service and request descriptions is almost the same, because they are all PSDL descriptions (1865 microseconds on average with less than 2% of standard deviation). Additionally, syntactic matching based on capability names (SLP advertisement and request) is the most efficient, which is due to the fact that there are less information to compare (only capability names). Finally, thanks to the optimized semantic matching defined in [8] semantic matching of capabilities (SAWSDL advertisement and request) is as efficient as syntactic matching of capabilities (WSDL, UPnP advertisements and requests). Regarding other semantic-based languages (e.g., WSMO, OWL-S), we expect the semantic matching of service capabilities to be performed as efficiently as SAWSDL matching as the descriptions are to be translated to PSDL.

Overall, it can be concluded that parsing and matching PSDL descriptions is also negligible when compared to the total discovery time (and in particular the processing time for SOAP communication).

Table 4. Syntactic-semantic parsing and matching processing time (micro-seconds)

		REQUESTED SERVICE							
		PSDL SLP		PSDL UPnP		PSDL WSDL		PSDL SAWSDL	
		Parsing	Matching	Parsing	Matching	Parsing	Matching	Parsing	Matching
ADVERTISED SERVICE	PSDL SLP	1798	**9.2**	1844	**9.0**	1832	**8.9**	1894	**11.3**
	PSDL UPnP	1907	**10.7**	1868	**18.0**	1859	**16.9**	1923	**18.8**
	PSDL WSDL	1882	**10.3**	1829	**18.1**	1822	**17.4**	1896	**19.0**
	PSDL SAWSDL	1888	**10.9**	1851	**19.3**	1841	**18.8**	1910	**21.3**

6 Conclusion

The ambient computing vision is increasingly enabled by the large success of wireless networks and devices. In ambient computing environments, heterogeneous software and hardware resources may be discovered and integrated transparently towards assisting the performance of users' daily tasks. An essential requirement towards the realization of such a vision is the availability of mechanisms enabling the discovery of resources that best fit the client applications' needs among the heterogeneous resources that populate the ambient computing environment and that may spontaneously appear/disappear into/from the network. Following on prior innovative solutions primarily addressing multi-protocols service discovery (MUSDAC) and semantic-awareness (EASY), we have introduced a conceptual model for service description as well as conformance relations for the efficient intermix of both syntactic-based and semantic-based SDPs. We have developed the PerSeSyn platform

that implements this model and conformance relations to provide a comprehensive solution for service matching and ranking in ambient computing environments. Performance evaluation of the current PerSeSyn prototype implementation validates our approach and demonstrates that semantics can be efficiently combined with legacy syntactic SDPs. Our future work includes the incorporation of multi-network interoperability into the PerSeSyn platform as well as dealing with service non-functional properties and in particular privacy-awareness towards a comprehensive service discovery platform for ambient computing environments. We further envision providing a complete performance evaluation of the overall resulting platform.

References

1. P. Grace, G. S. Blair, S. Samuel. "ReMMoC: A Reflective Middleware to Support Mobile Client Interoperability". In Proceedings of International Symposium on Distributed Objects and Applications, 2003.
2. P.-G. Raverdy, V. Issarny, R. Chibout, A. de La Chapelle. “A Multi-Protocol Approach to Service Discovery and Access in Pervasive Environments”. In *Proceedings of MOBIQUITOUS - The 3rd Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services*. July 2006, San Jose, CA, USA.
3. Jin Nakazawa, W. K Edwards, U. Ramachandran, and H, Tokuda, ”A Bridging Framework for Universal Interoperability in Pervasive Systems”, The 26th International Conference on Distributed Computing Systems (IEEE ICDCS 2006), Lisboa, Portugal, July 2006
4. T Koponen and T Virtanen. "Service Discovery: A Service Broker Approach" In Proceedings of the 37th Annual Hawaii International Conference on System Sciences, 2004.
5. Y-D Bromberg and V Issarny. "INDISS: Interoperable Discovery System for Networked Services". In *Proceedings of ACM/IFIP/USENIX 6th International* Middleware Conference (Middleware). 2005.
6. P-G Raverdy, O. Riva, A. de La Chapelle, R. Chibout, V. Issarny. “Efficient Context-aware Service Discovery in Multi-Protocol Pervasive Environments”. In *Proceedings of the 7th International Conference on Mobile Data Management (MDM'06)*. May 2006, Nara, Japan.
7. The DAML Services Coalition. “Bringing semantics to web services: The owl-s approach”. In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC’04)*, 2004.
8. S. Ben Mokhtar, D. Preuveneers, N. Georgantas, V. Issarny, Y. Berbers. EASY: Efficient SemAntic Service DiscoverY in Pervasive Computing Environments with QoS and Context Support. In Journal of System and Software, Volume 81 , Issue 5. 2008.
9. Seco, N., T. Veale & J. Hayes (2004). An intrinsic information content metric for semantic similarity in WordNet. In Proc. of ECAI-04, pp. 1089–1090.
10. Camara, J., Canal, C., Cubo, J., and Vallecillo, A. (2006). Formalizing wsbpel business processes using process algebra. ENTCS 154(1) :159–173.
11. Narayanan, S., McIlraith, S.: Simulation, Verification and Automated Composition of Web Services. In: WWW’02, ACM Press (2002) 77–88